**Java Object clone()**

1.The Java Object clone() method creates a shallow copy of the object.

2.Here, the shallow copy means it creates a new object and copies all the fields and methods associated with the object.

3.The syntax of the clone() method is:



4.The **java.lang.Cloneable interface** must be implemented by the class whose object clone we want to create.

## 5.clone() Parameters

The clone() method does not take any parameters.

## 6.clone() Return Values

## returns the copy of the object

throws CloneNotSupportedException if the object's class does not implement the Cloneable interface

### **7.Why use clone() method ?**

The **clone() method** saves the extra processing task for creating the exact copy of an object. If we perform it by using the new keyword, it will take a lot of processing time to be performed that is why we use object cloning.

### **8. Advantage of Object cloning**

* You don't need to write lengthy and repetitive codes. Just use an abstract class with a 4- or 5-line long clone() method.
* It is the easiest and most efficient way for copying objects, especially if we are applying it to an already developed or an old project. Just define a parent class, implement Cloneable in it, provide the definition of the clone() method and the task will be done.
* Clone() is the fastest way to copy array.

### **Disadvantage of Object cloning**

* To use the Object.clone() method, we have to change a lot of syntaxes to our code, like implementing a Cloneable interface, defining the clone() method and handling CloneNotSupportedException, and finally, calling Object.clone() etc.
* We have to implement cloneable interface while it doesn't have any methods in it. We just have to use it to tell the JVM that we can perform clone() on our object.
* Object.clone() is protected, so we have to provide our own clone() and indirectly call Object.clone() from it.
* Object.clone() doesn't invoke any constructor so we don't have any control over object construction.
* If you want to write a clone method in a child class then all of its superclasses should define the clone() method in them or inherit it from another parent class. Otherwise, the super.clone() chain will fail.
* Object.clone() supports only shallow copying but we will need to override it if we need deep cloning.

**Example**

class Main implements Cloneable {

// declare variables

String name;

int version;

public static void main(String[] args) {

// create an object of Main class

Main obj1 = new Main();

// initialize name and version using obj1

obj1.name = "Java";

obj1.version = 14;

// print variable

System.out.println(obj1.name); // Java

System.out.println(obj1.version); // 14

try {

// create clone of obj1

Main obj2 = (Main)obj1.clone();

// print the variables using obj2

System.out.println(obj2.name); // Java

System.out.println(obj2.version); // 14

}

catch (Exception e) {

System.out.println(e);

}

}

}

## **Example 2: Changing value using cloned object**

class Main implements Cloneable {

// declare variables

String name;

int version;

public static void main(String[] args) {

// create an object of Main class

Main obj1 = new Main();

// initialize name and version using obj1

obj1.name = "Java";

obj1.version = 14;

// print variable

System.out.println(obj1.name); // Java

System.out.println(obj1.version); // 14

try {

// create a clone of obj1

Main obj2 = (Main)obj1.clone();

// print the variables using obj2

System.out.println(obj2.name); // Java

System.out.println(obj2.version); // 14

// changing value of name

// using obj2

obj2.name = "Python";

System.out.println(obj2.name); // Python

// check if value associated

// with obj1 is changed

System.out.println(obj1.name); // Java

}

catch (Exception e) {

System.out.println(e);

}

}

}